REMARKS/ARGUMENTS

Claims 1-24 are pending in the present application. Claims 1-16, 19 and 21-24 have been amended herewith. Reconsideration of the claims is respectfully requested.

I. 35 U.S.C. § 103, Obviousness

Claims 1, 6, 9, 14, 17 and 22 stand rejected under 35 U.S.C. § 103 as being unpatentable over Intel *("Itanium™ Processor Floating-point Software Assistance and Floating-point Exception Handling." Intel Corp., January 2000)* in view of Traut (US Patent No. 7,085,705*)*. This rejection is respectfully traversed.

With respect to Claim 1 (and dependent Claim 6), such claim has been amended to include the features previously recited in dependent Claims 2 and 4. Applicants traverse the rejection of amended Claim 1 (and dependent Claim 6) for reasons given below in the discussion pertaining to the rejection of Claim 4.

Further with respect to Claim 6, such claim has been amended to further define characteristics of the symmetric multi-processor system. None of the cited references teach or suggest such a symmetric multiprocessor system. In rejecting Claim 6, the Examiner cites Intel's teaching at page 7-2, paragraph 4 as teaching the claimed symmetric data processing system. Applicants urge that this cited passage briefly alludes to synchronizing processors in a multiprocessor system, but makes no mention or suggestion that such multiprocessing system is a symmetric multi-processing system, as claimed. Accordingly, the Examiner has failed to establish a prima facie showing of obviousness with respect to Claim 6[1], and accordingly such claim has been erroneously rejected[2].

Applicants traverse the rejection of Claim 9 (and dependent Claim 14) for similar reasons to those given with respect to Claim 1.

Applicants further traverse the rejection of Claim 14 for similar reasons to the further reasons given with respect to Claim 6.

With respect to Claim 17 (and dependent Claim 22), it is urged that none of the cited references teach or suggest the claimed unitary computer program product that comprises all of the recited first instructions, second instructions, third instructions and fourth instructions. In rejecting Claim 17, the

---

[1] To establish prima facie obviousness of a claimed invention, all of the claim limitations must be taught or suggested by the prior art. MPEP 2143.03. *See also, In re Royka*, 490 F.2d 580 (C.C.P.A. 1974).

[2] If the examiner fails to establish a prima facie case, the rejection is improper and will be overturned. *In re Fine*, 837 F.2d 1071, 1074, 5 USPQ2d 1596, 1598 (Fed. Cir. 1988).

Examiner cites different passages from different references in order to singularly reject each individual claimed element, but provides no teaching, suggestion or other rationale as to why it would have been obvious to include all of the first instructions, second instructions, third instructions and fourth instructions on the same computer recordable-type medium, as per the features of Claim 17. Accordingly, the Examiner has failed to establish a prima facie showing of obviousness with respect to Claim 17 (and dependent Claim 22), and accordingly such claim has been erroneously rejected.

Therefore, the rejection of Claims 1, 6, 9, 14, 17 and 22 under 35 U.S.C. § 103 has been overcome.

II. 35 U.S.C. § 103, Obviousness

Claims 2, 4, 5, 7, 8, 10, 12, 13, 15, 16, 18, 20, 21, 23 and 24 stand rejected under 35 U.S.C. § 103 as being unpatentable over Intel in view of Traut and in further view of Jones *(Jones, Steve. "Using spinlocks in a symmetric multiprocessing environment." Tech Specialist, v2, n10, pg 15(6), Oct 1991).* This rejection is respectfully traversed.

Applicants initially traverse the rejection of Claim 2 for reasons given with respect to Claim 1 (of which Claim 2 depends upon).

Further with respect to Claim 2, this amended claim now recites "wherein the saving, replacing, and restoring steps are conditionally performed only if the data processing system is in a debug mode as determined by a debug flag maintained by the firmware". It is urged that none of the cited references teach or suggest such conditional performance of the saving, replacing and restoring steps (as depicted in the preferred embodiment in Figure 4, blocks 402, 406, 408, 412 and 414). This claimed feature advantageously provides for providing a special debug mode for debugging the firmware itself (Specification page 14, lines 20-22 and page 16, lines 24-27). Thus, it is further urged that amended Claim 2 is not obvious in view of the cited references in addition to not being obvious due to its dependency on amended Claim 1.

With respect to previous Claim 4, whose features are now a part of amended Claim 1, it is urged that none of the cited references teach or suggest the claimed feature of "wherein context for the processor in the slave loop is stored to form a stored context". In rejecting this aspect of Claim 4, the Examiner states:

> "The use of a spinlock to pause the execution of a *processor inherently requires that the state of the processor be saved* because, while the processor keeps checking the state of the lock, it must use registers (memory locations local to the processor that are used to perform operations) previously occupied by the previous task being executed. The

contents of the registers that correspond to the information relating to the exception must be stored in RAM or some other storage medium to allow the processor to continue with that exception, once the lock is held by that processor. Once that lock is held, the *processor must inherently load the saved context back* into the registers in order to continue its previous operation." (emphasis added by Applicants)

Applicants urge error in such inherency assertion. "Inherency . . . may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient." *In re Oelrich*, 666 F.2d 578, 581, 212 USPQ 323, 326 (CCPA 1981) (quoting *Hansgirg v. Kemmer*, 102 F.2d 212, 214, 40 USPQ 665, 667 (CCPA 1939)). "To establish inherency," the Federal Circuit recently stated, "the extrinsic evidence '**must make clear that the missing descriptive matter is necessarily present in the thing described in the reference**, and that it would be so recognized by persons of ordinary skill.'" *In re Robertson*, 169 F.3d 743, 745 [49 USPQ2d 1949] (Fed. Cir. 1999) (emphasis added by Applicants); see also *Continental Can Co. U.S.A., Inc. v. Monsanto Co.*, 948 F.2d 1264, 1268 [20 USPQ2d 1746] (Fed. Cir. 1991). Such inherency may not be established by "'probabilities or possibilities.'" *Continental Can*, 948 F.2d at 1269 (quoting In re Oelrich, 666 F.2d 578, 581 [212 USPQ 323] (C.C.P.A. 1981)). Applicants will now show that the missing matter alleged to be inherent is in fact not inherent as it is not necessarily present in the thing described in the reference.

As evidenced by Attachment I in the previous response filed by Applicants on November 15, 2006, a spinlock is a lock where the thread simply waits in a loop ("spins") repeatedly checking until the lock becomes available. As the thread remains active but isn't performing a useful task, the use of such a lock is a kind of busy waiting. **Spinlocks are efficient if threads are only likely to be blocked for a short period of time, as they *avoid overhead from operating system process re-scheduling or context switching***. For this reason, spinlocks are often used inside operating system kernels. Thus, it is shown that context switching by a processor is not necessarily present when using a spinlock, as alleged by the Examiner, and in fact such **context switching is not done** when using a spinlock as described in this Attachment I.

To further establish that it is not inherent to perform a context switch when using a spinlock, Applicants have attached hereto as Attachment A an article entitled "User-Level Spin Locks", by Gert Boddaert. As stated on page 3 of this article:

"4.1 When should you use a Spin Lock?

Only when it makes sense not to do a context switch in the event of a locked resource"

This shows that it is not in fact inherent to perform a context switch when using a spinlock, and this article – just like the previous article that was brought to the Examiner's attention in the last response – *expressly teaches away from performing a context switch when using a spinlock.*

Perhaps the Examiner's mischaracterization of spinlocks requiring context switching is based on an erroneous assumption regarding the presently claimed operating environment – that of a multi-processor system. The erroneous assumption is somewhat evidenced by the following Examiner comment found on the bottom of page 10 and extending to the top of page 11 of the present Office Action dated February 6, 2007. There, the Examiner states:

> "The use of a spinlock to pause the execution of a *processor inherently requires that the state of the processor be saved* because, while the processor keeps checking the state of the lock, **it must use registers (memory locations local to the processor that are used to perform operations) previously occupied by the previous task being executed.** The contents of the registers that correspond to the information relating to the exception must be stored in RAM or some other storage medium to allow the processor to continue with that exception, once the lock is held by that processor." (page 5 of the present Office Action)
>
> "**When replacing one process with another in an executing processor,** the state of the previous processes must be saved in order to allow continued execution of that preempted process at a later time. Therefore, Examiners (sic) contention of inherency in the storage of state data in regard to Claims 4 and 5 is valid" (page 11 of the present Office Action).

While Applicants offer no opinion as to the validity of these statements in a single processor system, these assertions are *not true in a multiprocessing system* such as is claimed in Claim 1 (which recites both a 'processor' and an 'associated processor'). Because there are *multiple processors* available for performing code execution, there is no requirement that a processor spinning on a lock – which is what the Examiner alleges to be taught by the cited references – must perform a context saving operation or a context switching operation *as there is another processor* that is capable of performing the operations for which the first (spinning) processor is awaiting completion of. Restated, there is no requirement for a processor spinning on a lock to perform a context switch as there are other processing resources that are available for executing code/processes and therefore no context switching is required for the spinning processor. The two articles that have been cited by Applicants (one in the previous response (Attachment

I) and one in the current response (Attachment A)) conclusively establish that *spinlocks are specifically used in environments when **no** **context switching** is desired.* Thus, the Examiner's inherency assertion regarding spinlocks and context switching are clearly erroneous.

Therefore, the missing claimed features identified above as not be expressly taught or suggested by the cited references *are also not inherent*[3]. Thus, previous Claim 4 (and dependent Claim 5), whose features are now a part of amended Claim 1, has been erroneously rejected as there are claimed features not taught (either expressly or inherently) or suggested by the cited references. It is thus urged that Claim 1 (and dependent Claims 2, 4 and 5), which has been amended to include features previously recited in Claims 2 and 4, is not obvious in view of the cited references as there are missing claimed features not taught or suggested by any of the cited references.

With respect to newly amended Claim 4 (and dependent Claim 5), this amended claim now recites "wherein the placing step is conditionally performed only if a processor identification number for the processor does not match an associated processor identification number for the associated processor". It is urged that none of the cited references teach or suggest such conditional performance of the placing step (as depicted in Figure 5, blocks 504 and 506). This claimed feature advantageously provides for different code execution processing depending upon which processor (the processor or the associated processor) is accessing an exception vector in order to allow the associated processor to process interrupts notwithstanding that the exception vector has been replaced. Thus, it is further urged that amended Claim 4 (and dependent Claim 5) is not obvious in view of the cited references in addition to not being obvious due to its dependency on amended Claim 1.

With respect to Claim 7 (and dependent Claim 8), such claim has been amended to specify that each of the claimed steps recited therein are performed by the firmware itself. It is urged that none of the cited references teach or suggest such a firmware operation. In rejecting Claim 7, the Examiner states that these steps are taught by Intel page 1-2, paragraphs 4-6; Intel page 2-2, paragraph 4-7; and Traut col. 6, line 44 – col. 7, line 45. Applicants urge that these passages describe a plurality of different and disjoint processes for performing various actions, including (i) a kernel floating-point trap handler (Intel page 1-2, paragraph 6), (ii) an exception handler (Intel page 1-2, paragraph 6), and (iii) an emulation library handler (Intel page 1-2, paragraph 6), as depicted at Intel Figure 1-1 on page 1-3. There is no mention of a single, unitary firmware that performs all of the steps recited in Claim 7, in either the cited Intel reference or the cited Traut reference. For example, none of the cited references teach or suggest

[3] "To establish inherency," the Federal Circuit recently stated, "the extrinsic evidence `must make clear that the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill.'" *In re Robertson*, supra; see also *Continental Can Co. U.S.A., Inc. v. Monsanto Co.*, supra. Such inherency may not be established by "`probabilities or possibilities.'" *Continental Can*, supra (quoting In re Oelrich, supra).

firmware that replaces an exception with substitute code. Thus, it is urged that amended Claim 7 (and dependent Claim 8) is not obvious in view of the cited references as there are missing claimed features not taught or suggested by such cited references.

Further with respect to Claim 8, such claim recites "wherein the processors are suspended by placing the processors in a slave loop by the firmware only if a debug mode for the firmware is enabled". It is urged that none of the cited references teach or suggest firmware, that replaces an exception vector with substitute code, also conditionally places processors in a slave loop. In rejecting Claim 8, the Examiner states that the spinlock as described by Jones teaches a slave loop to pause execution of a BIOS routine (the BIOS routine being a form of firmware). Applicants urge that since per this scenario the firmware itself is paused by the spinlock, and therefore such firmware could not operate to place a processor in a slave loop as such (alleged) firmware is paused by a separate spinlock. Thus, it is further urged that amended Claim 8 is not obvious in view of the cited references as there are additional missing claimed features not taught or suggested by such cited references.

Applicants traverse the rejection of Claims 10, 12 and 13 for similar reasons to those given above with respect to Claims 2, 4 and 5.

With respect to Claim 15 (and dependent Claim 16), such claim has been amended to recite details of the multiprocessing system and related firmware. It is urged that amended Claim 15 is not obvious in view of the cited references for similar reasons to those given above with respect to Claims 6 and 7.

Further with respect to Claim 16, it is urged that none of the cited references teach or suggest that the processors are suspended by placing the processors in a slave loop by the firmware only if a debug mode for the firmware is enabled, as none of the cited references teach or suggest a firmware debug mode. Thus, it is further urged that Claim 16 is not obvious in view of the cited references as there are additional claimed features not taught or suggested by the cited references.

Applicants traverse the rejection of Claim 18 for similar reasons to those given above with respect to Claims 17), and urge that the additional cited IBM reference does not overcome the teaching/suggestion deficiencies identified above with respect to amended Claim 17.

Applicants traverse the rejection of Claim 20 (and dependent Claim 21) for similar reasons to those given above with respect to amended Claim 1, and for reasons given above with respect to Claim 17 (of which Claim 20 ultimately depends upon).

With respect to Claim 23 (and dependent Claim 24), such claim has been amended to recite details of the substitute code. It is urged that amended Claim 23 is not obvious in view of the cited references as none of the cited references teach or suggest debug code for debugging the computer program product itself, where such debug code replaces an exception vector.

Further with respect to Claim 24, it is urged that none of the cited references teach or suggest that the processors are suspended by placing the processors in a slave loop by the firmware only if a debug mode for the computer program product is enabled, as none of the cited references teach or suggest a computer program product debug mode. Thus, it is further urged that Claim 24 is not obvious in view of the cited references as there are additional claimed features not taught or suggested by the cited references.

Therefore, the rejection of Claims 2, 4, 5, 7, 8, 10, 12, 13, 15, 16, 18, 20, 21, 23 and 24 under 35 U.S.C. § 103 has been overcome.

III. 35 U.S.C. § 103, Obviousness

Claims 3, 11 and 19 stand rejected under 35 U.S.C. § 103 as being unpatentable over Intel in view of Traut and in further view of Jones and in further view of IBM *(IBM Technical Bulletin: NNRD447149. "Method to Prevent Multiple Processes After Taking Exceptions To Enter Open Firmware In a Symmetrical Multiprocessor Machine", Published 01 July 2001)*. This rejection is respectfully traversed.

Applicants traverse the rejection of Claim 3 (and similarly for Claim 11) for reasons given above with respect to amended Claim 1 (of which Claim 3 depends upon), and urge that the additional cited Jones and IBM references do not overcome the teaching/suggestion deficiencies identified above with respect to amended Claim 1.

Applicants initially traverse the rejection of Claim 19 for reasons given above with respect to Claim 17 (of which Claim 19 depends upon), and urge that the additional cited Jones and IBM references do not overcome the teaching/suggestion deficiencies identified above with respect to Claim 17.

Applicants further traverse the rejection of Claim 19 for similar reasons to the further reasons given above with respect to Claim 4), and urge that the additional cited Jones and IBM references do not overcome the teaching/suggestion deficiencies identified above with respect to Claim 4.

Therefore, the rejection of Claims 3, 11 and 19 under 35 U.S.C. § 103 has been overcome.

IV. Conclusion

It is respectfully urged that the subject application is patentable over the cited references and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: May 7, 2007

Respectfully submitted,

/Wayne P. Bailey/

Wayne P. Bailey
Reg. No. 34,289
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants